Introduction to Grafana

1. **What is Grafana?**
    - Grafana is an open-source platform used for monitoring and observability, specifically for visualising time-series data.
    - It works with various data sources (e.g., Prometheus, Graphite, Elasticsearch) to provide real-time dashboards and alerts.

- Grafana is often used with Prometheus to visualise metrics from applications and infrastructure.

2. **Why Grafana for SRE?**
   - You can create dashboards with customisable panels to display key metrics in different formats (graphs, heatmaps, gauges, etc.).
   - Grafana integrates with multiple backends, including Prometheus, InfluxDB, MySQL, and AWS CloudWatch, making it ideal for complex systems.
   - In addition to visualisation, Grafana allows you to set up alerts based on predefined thresholds (e.g., CPU utilization > 80%) and send notifications via Slack, PagerDuty, or email.
3. **Grafana's Role in Observability:**
   - Grafana is often the front-end tool for monitoring and observability, offering a way to interpret data coming from Prometheus or other sources.
   - It provides detailed insights into system performance by visually representing various metrics over time, helping SRE teams quickly detect and diagnose issues.

▼ Setting up Grafana with Prometheus

Step 1: Install Grafana

- **Install via Docker:**

```
docker run -d -p 3000:3000 --name=grafana grafana/gra
fana
```

- **Install via Package Manager**:
  - For Ubuntu: `sudo apt-get install -y grafana`
  - For macOS (Homebrew): `brew install grafan`

Step 2: Add Prometheus as a Data Source in Grafana

- Log into Grafana by visiting `http://localhost:3000` (default username/password: `admin` / `admin`).
- Navigate to **Configuration** > **Data Sources** > **Add Data Source**.

- Select **Prometheus** as the data source.
- Configure the URL to point to your Prometheus server, typically `http://host.docker.internal:9090` if Prometheus is running in Docker and Grafana needs to connect to it from the host, or `http://prometheus:9090` if on the same Docker network.

Step 3: Create a Dashboard for Spring Boot Metrics

Since we're only using metrics directly available from the Spring Boot application, we'll focus on those without relying on Node Exporter.

1. **Add a Visualisation Panel**:
   - Click on **+ Add visualisation** to open the panel editor.
   - This will allow you to select the data source, define the query, and customise the display.
2. **Select Prometheus as the Data Source**:
   - In the panel editor, locate the **Data Source** dropdown at the top of the query editor. Select **Prometheus**.
3. **Create a Query with Available Spring Boot Metrics**:
   - Use queries based on the metrics available in your Spring Boot application's `/actuator/prometheus` endpoint. Here are some examples:
     - **Application Uptime**:

       ```
       process_uptime_seconds
       ```

     - **Application Start Time**:

       ```
       application_started_time_seconds
       ```

     - **Memory Usage (JVM)**:

       ```
       jvm_memory_used_bytes{area="heap"}
       ```

     - **CPU Usage (Process CPU Time)**:

       ```
       process_cpu_time_ns_total
       ```

- **HTTP Request Metrics** (if active requests are available):

```
http_server_requests_active_seconds_max
```

- These queries use Spring Boot's native metrics. As you type, you should see real-time feedback from Prometheus if it's connected correctly.

4. **Customise the Panel**:
   - Select the visualisation type (e.g., **Time Series** for most metrics, **Gauge** for memory or uptime metrics).
   - Customise appearance, title, and other display options to make the panel informative and clear.
5. **Save the Panel**:
   - After setting up the panel, click **Apply** in the top-right corner to save it to your dashboard.
6. **Repeat to Add Additional Panels**:
   - Add more panels to visualise different metrics as needed, like memory usage, CPU usage, uptime, etc.
   - Save the entire dashboard by clicking **Save dashboard** (disk icon) in the top-right corner.

Step 4: Set Up Alerts (Optional)

If you'd like to monitor critical metrics, Grafana allows you to set alerts:

- Alerts can be added to any panel by configuring thresholds for your metrics (e.g., high memory usage).
- Alerts can send notifications to **Slack**, **PagerDuty**, **email**, or other systems, allowing proactive monitoring.

▼ Additional Exporters

While your current setup focuses on metrics provided by Spring Boot, you can extend your monitoring capabilities by using various exporters that provide detailed system and application-level metrics. Below are some popular exporters that can integrate seamlessly with Prometheus and Grafana.

▼ **Node Exporter** (for System-Level Metrics)

Node Exporter is a widely used exporter for gathering hardware and OS-level metrics. It collects data on CPU, memory, disk I/O, filesystem usage, and more. It's particularly useful for tracking the health and performance of the underlying infrastructure.

- **Installation via Docker**:

```
docker run -d -p 9100:9100 --name=node_exporter pr
om/node-exporter
```

- **Example Metrics Available**:
  - **CPU Usage**: `rate(node_cpu_seconds_total{mode="user"}[5m])`
  - **Memory Usage**: `(node_memory_MemAvailable_bytes / node_memory_MemTotal_bytes) * 100`
  - **Disk Space**: `node_filesystem_avail_bytes`
- **Prometheus Configuration**:
  - Add Node Exporter as a scrape target in `prometheus.yml`:
- **Use in Grafana**: Once Node Exporter metrics are available in Prometheus, you can create panels in Grafana to monitor CPU, memory, and disk usage.

▼ **cAdvisor (for Container Metrics)**

cAdvisor is a tool developed by Google for container monitoring. It provides insights into the performance of Docker containers, including CPU and memory usage, network traffic, and disk I/O.

- **Installation via Docker**:

```
docker run -d --name=cadvisor -p 8081:8080 \
  --volume=/:/rootfs:ro \
  --volume=/var/run:/var/run:ro \
  --volume=/sys:/sys:ro \
  --volume=/var/lib/docker/:/var/lib/docker:ro \
  gcr.io/cadvisor/cadvisor
```

- **Example Metrics Available**:

    - **Container CPU Usage**: `container_cpu_usage_seconds_total`
    - **Container Memory Usage**: `container_memory_usage_bytes`
    - **Container Network Traffic**: `container_network_receive_bytes_total`
  - **Prometheus Configuration**:
    - Add cAdvisor as a scrape target in `prometheus.yml`:

```
- job_name: 'cadvisor'
  static_configs:
    - targets: ['localhost:8081']
```

  - **Use in Grafana**: cAdvisor metrics are especially useful for monitoring resource usage across multiple containers and spotting bottlenecks.

▼ **Database Exporters (e.g., PostgreSQL, MySQL)**

Database exporters allow you to monitor the health and performance of databases, tracking query performance, connection stats, cache usage, and more.

- **PostgreSQL Exporter**:
  - **Installation via Docker**:

```
docker run -d -p 9187:9187 --name=postgres_expo
rter \
  -e DATA_SOURCE_NAME="postgresql://user:passwo
rd@hostname:5432/dbname" \
  quay.io/prometheuscommunity/postgres-exporter
```

  - **Example Metrics**:
    - **Active Connections**: `pg_stat_activity_count`
    - **Cache Hit Rate**: `pg_stat_cache_hits_ratio`
    - **Slow Queries**: `pg_stat_activity_duration`
- **MySQL Exporter**:
  - **Installation via Docker**:

```
bash
Copy code
docker run -d -p 9104:9104 --name=mysql_exporte
r \
  -e DATA_SOURCE_NAME="user:password@(hostname:
3306)/" \
  prom/mysqld-exporter
```

    - **Example Metrics**:
        - **Queries Per Second**: `mysql_global_status_queries`
        - **Connection Count**: `mysql_global_status_threads_connected`
        - **Buffer Pool Usage**: `mysql_global_status_innodb_buffer_pool_bytes_data`

- **Prometheus Configuration**:
    - Add the database exporter as a scrape target in `prometheus.yml`:

```
yaml
Copy code
- job_name: 'postgres-exporter'
  static_configs:
    - targets: ['localhost:9187']
```

```
yaml
Copy code
- job_name: 'mysql-exporter'
  static_configs:
    - targets: ['localhost:9104']
```

- **Use in Grafana**: Database metrics are invaluable for monitoring query performance and ensuring database health under varying load conditions.

▼ **Blackbox Exporter** (for Network and Endpoint Monitoring)

Blackbox Exporter allows you to monitor endpoints using HTTP, TCP, ICMP, and DNS checks. This is useful for ensuring that critical endpoints are reachable and responding as expected.

- **Installation via Docker**:

```
docker run -d -p 9115:9115 --name=blackbox_exporte
r prom/blackbox-exporter
```

- **Example Checks Available**:
  - **HTTP Status Check**: Ensure that a website is reachable and responding with a 200 status.
  - **TCP Port Check**: Verify that a specific TCP port on a host is open and reachable.
- **Prometheus Configuration**:
  - Configure Blackbox Exporter as a scrape target in `prometheus.yml`:

```
- job_name: 'blackbox'
  metrics_path: /probe
  params:
    module: [http_2xx]  # Module for HTTP 200 O
K checks
  static_configs:
    - targets:
      - http://example.com
```

- **Use in Grafana**: Network and endpoint checks can help ensure service uptime and quickly detect outages or connectivity issues.

▼ Best Practices

1. **Clear Visual Hierarchy:**
   - Use Grafana's layout options to arrange panels in a clear, logical manner. Prioritize critical metrics at the top of the dashboard.
2. **Use Colours Sparingly:**

- Colour coding can be helpful for quick interpretation, but overuse of colors can be distracting. Use color changes for alerts or to emphasize abnormal behavior.

3. **Monitor the Right Metrics:**
    - Focus on key performance indicators (KPIs) like latency, error rates, traffic, and saturation. These metrics are usually aligned with SLOs and SLIs.
4. **Aggregate Similar Data:**
    - Use Grafana's ability to aggregate metrics from multiple sources to present an overall picture of system health. Avoid showing individual service metrics unless necessary.
5. **Annotations for Context:**
    - Grafana allows you to add annotations to your graphs to highlight important events (e.g., deployments or system failures) that provide context to your metrics.
